Blockchain

CS642: Computer Security

Outline

- Vision of digital cash
  - Centralized models
  - Decentralized models
- Block chain / distributed ledger
- Bitcoin details
- Ethereum details

A Brave New World - The Vision of David Chaum

INTRODUCTION

Automation of the way we pay for goods and services is already underway, as can be seen by the variety and growth of electronic banking services available to consumers. The ultimate structure of the new electronic payments system may have a substantial impact on personal privacy as well as on the nature and extent of criminal use of payments. Ideally a new payments system should address both of these seemingly conflicting sets of concerns.

On the one hand, knowledge by a third party of the payee, amount, and time of payment for every transaction made by an individual can reveal a great deal about the individual's whereabouts, associations and lifestyle. For example, consider payments for such things as transportation, hotels, restaurants, movies, theater, lectures, food, pharmaceuticals, alcohol, books, periodicals, dues, religious and political contributions.

On the other hand, an anonymous payments systems like bank notes and coins suffers from lack of controls and security. For example, consider problems such as lack of proof of payment, theft of payments media, and black payments for bribes, tax evasion, and black markets.

A Brave New World - The Vision of David Chaum

Basically ...

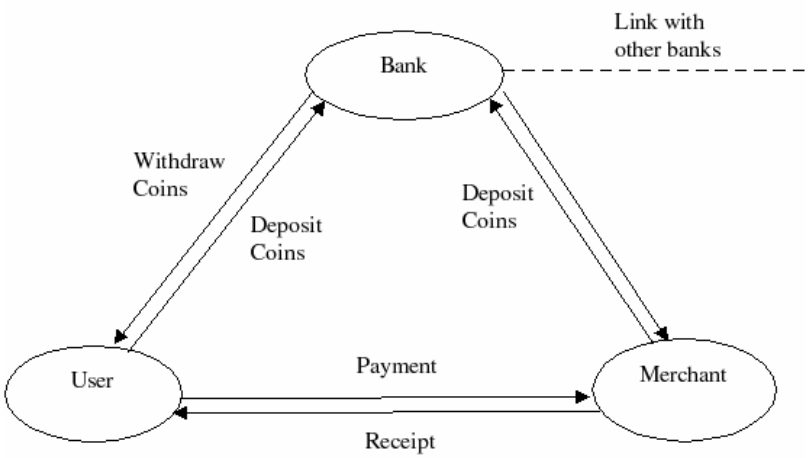
- Electronic payment systems suffer from loss of privacy and cumbersome trust on single entities.
- Privacy protection, however, encounters issues of security and safety of data.

Digital Payments

- Credit card approach
  - Vendors establish account with payment processor (e.g., PayPal)
  - User provides credit card number to vendor
  - During transaction, vendor contacts PayPal, which contacts Visa, which contacts a bank to authorize payment
  - Later, sends settlement request to transfer money to seller account
- Properties:
  - Expensive to scale: must handle billions of payments
  - Supports small number of credit cards
  - Identifies buyer (CC#), seller (account #)

Digital cash

- Goal: untraceable money transfers
- Idea: bank issues digital coins
  - Certificates signed by bank with a value
  - User can transfer them as payment to a merchant
  - Merchant can redeem them at a bank for money



Goofy can create new coins

signed by pk_{Goofy}

CreateCoin [uniqueCoinID]

A coin's owner can spend it.

signed by pk_{Goofy}

Pay to pk_{Alice} : H( )

signed by pk_{Goofy}

CreateCoin [uniqueCoinID]

The recipient can pass on the coin again.

signed by pk_{Alice}

Pay to pk_{Bob} : H( )

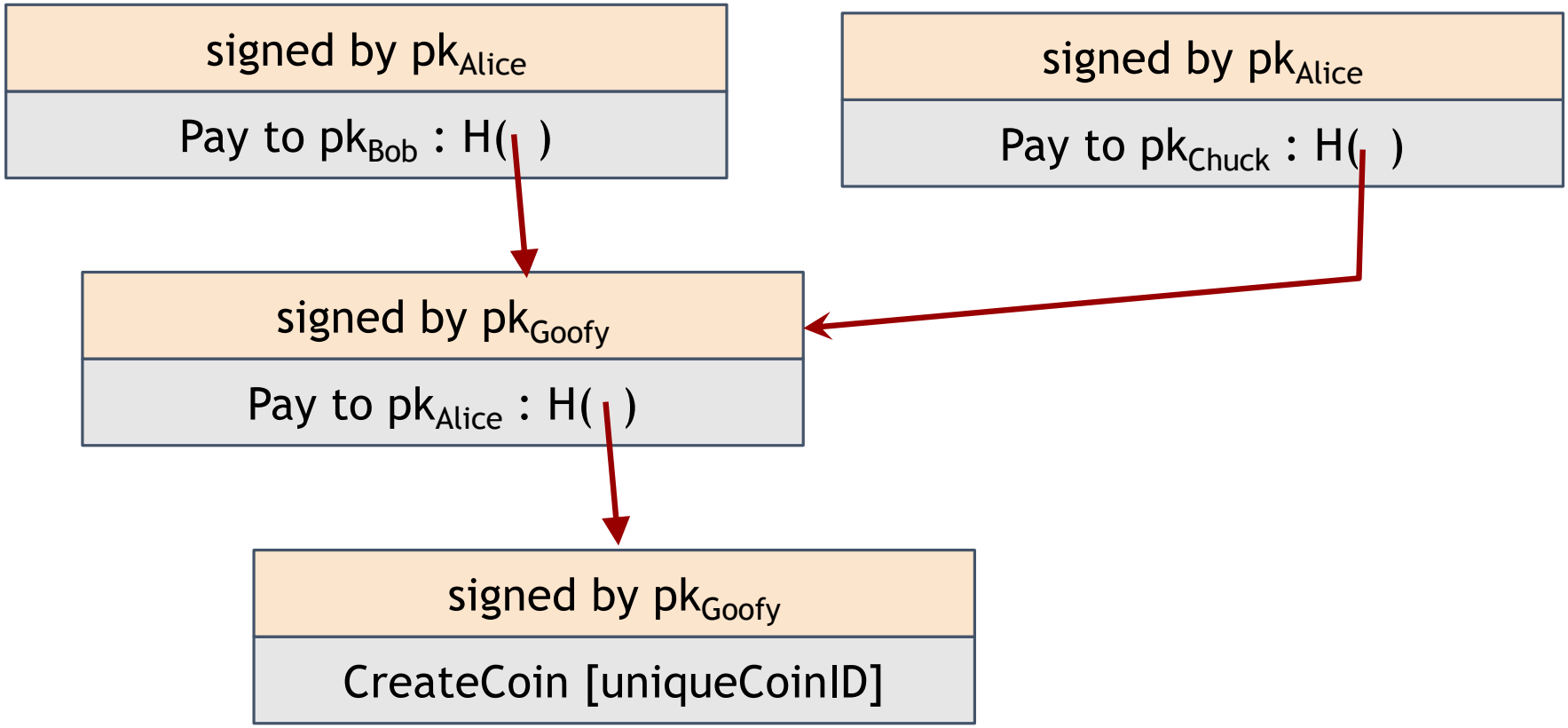
signed by pk_{Goofy}

Pay to pk_{Alice} : H( )

signed by pk_{Goofy}

CreateCoin [uniqueCoinID]

double-spending attack



Nick Szabo [1998]

Bit gold

A long time ago I hit upon the idea of bit gold. The problem, in a nutshell, is that our money currently depends on trust in a third party for its value. As many inflationary and hyperinflationary episodes during the 20th century demonstrated, this is not an ideal state of affairs. Similarly, private bank note issue, while it had various advantages as well as disadvantages, similarly depended on a trusted third party.

Thus, it would be very nice if there were a protocol whereby unforgeably costly bits could be created online with minimal dependence on trusted third parties, and then securely stored, transferred, and assayed with similar minimal trust. Bit gold.

My proposal for bit gold is based on computing a string of bits from a string of challenge bits, using functions called variously "client puzzle function," "proof of work function," or "secure benchmark function.". The resulting string of bits is the proof of work. Where a one-way function is prohibitively difficult to compute backwards, a secure benchmark function ideally comes with a specific cost, measured in compute cycles, to compute backwards.

http://unenumerated.blogspot.com/2005/12/bit-gold.html

Satoshi Nakamoto and the Anon Post [2008]

Bitcoin open source implementation of P2P currency

Posted by Satoshi Nakamoto on February 11, 2009 at 22:27

View Discussions

I've developed a new open source P2P e-cash system called Bitcoin. It's completely decentralized, with no central server or trusted parties, because everything is based on crypto proof instead of trust. Give it a try, or take a look at the screenshots and design paper:

Download Bitcoin v0.1 at http://www.bitcoin.org

Satoshi Nakamoto and the Anon Post [2008]

> The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust. Banks must be trusted to hold our money and transfer it electronically, but they lend it out in waves of credit bubbles with barely a fraction in reserve. We have to trust them with our privacy, trust them not to let identity thieves drain our accounts. Their massive overhead costs make micropayments impossible.

Satoshi Nakamoto and the Anon Post [2008]

A generation ago, multi-user time-sharing computer systems had a similar problem. Before strong encryption, users had to rely on password protection to secure their files, placing trust in the system administrator to keep their information private. Privacy could always be overridden by the admin based on his judgment call weighing the principle of privacy against other concerns, or at the behest of his superiors. Then strong encryption became available to the masses, and trust was no longer required. Data could be secured in a way that was physically impossible for others to access, no matter for what reason, no matter how good the excuse, no matter what.

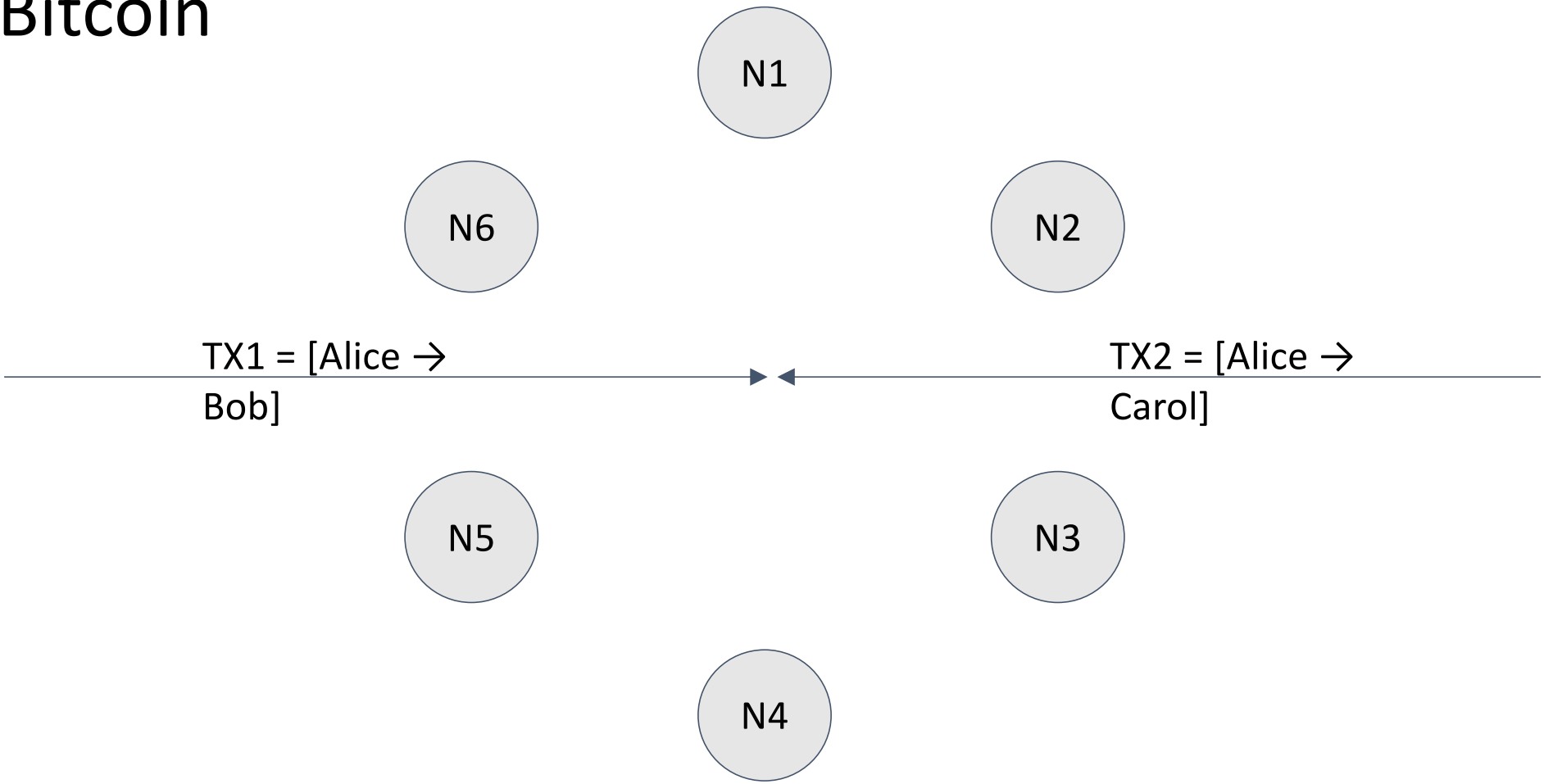
It's time we had the same thing for money. With e-currency based on cryptographic proof, without the need to trust a third party middleman, money can be secure and transactions effortless.

Goals

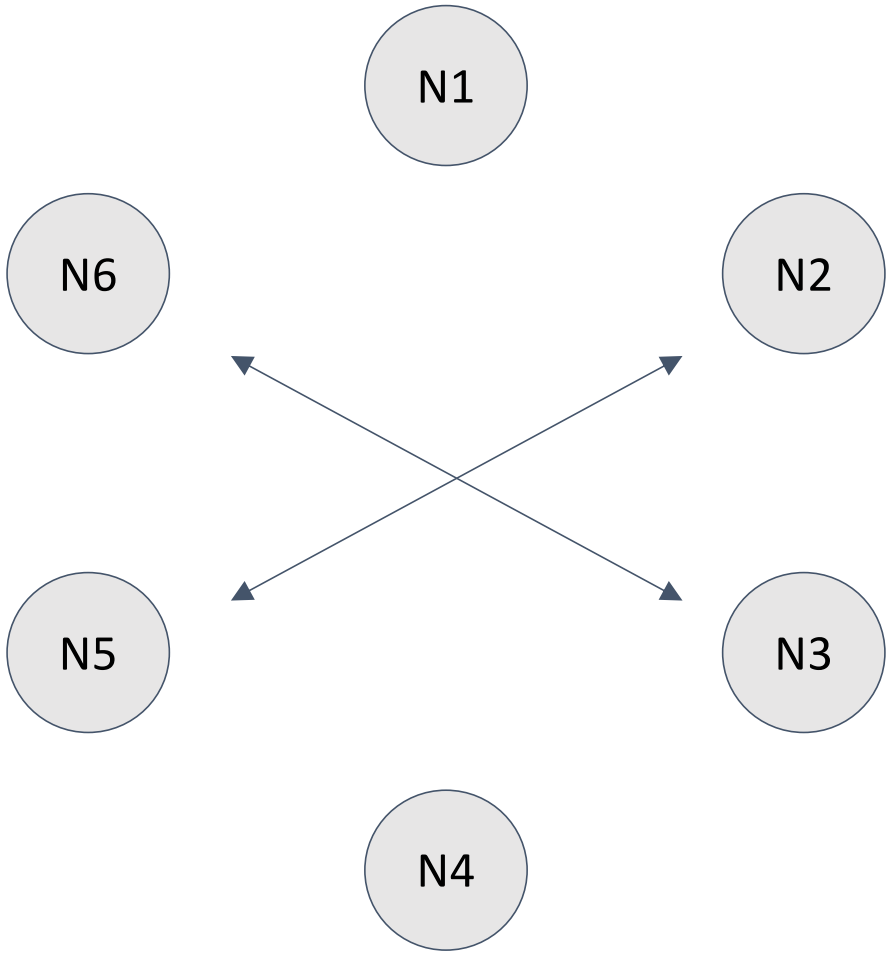
An electronic payment system:

- Guarantees safety of transactions, protects against double spends, gives full freedom to owners.
- Yet no central trusted authority, no reliance on quorum since identities are not known.
- Prevent compromise through **proof of work** – doing lots of computations

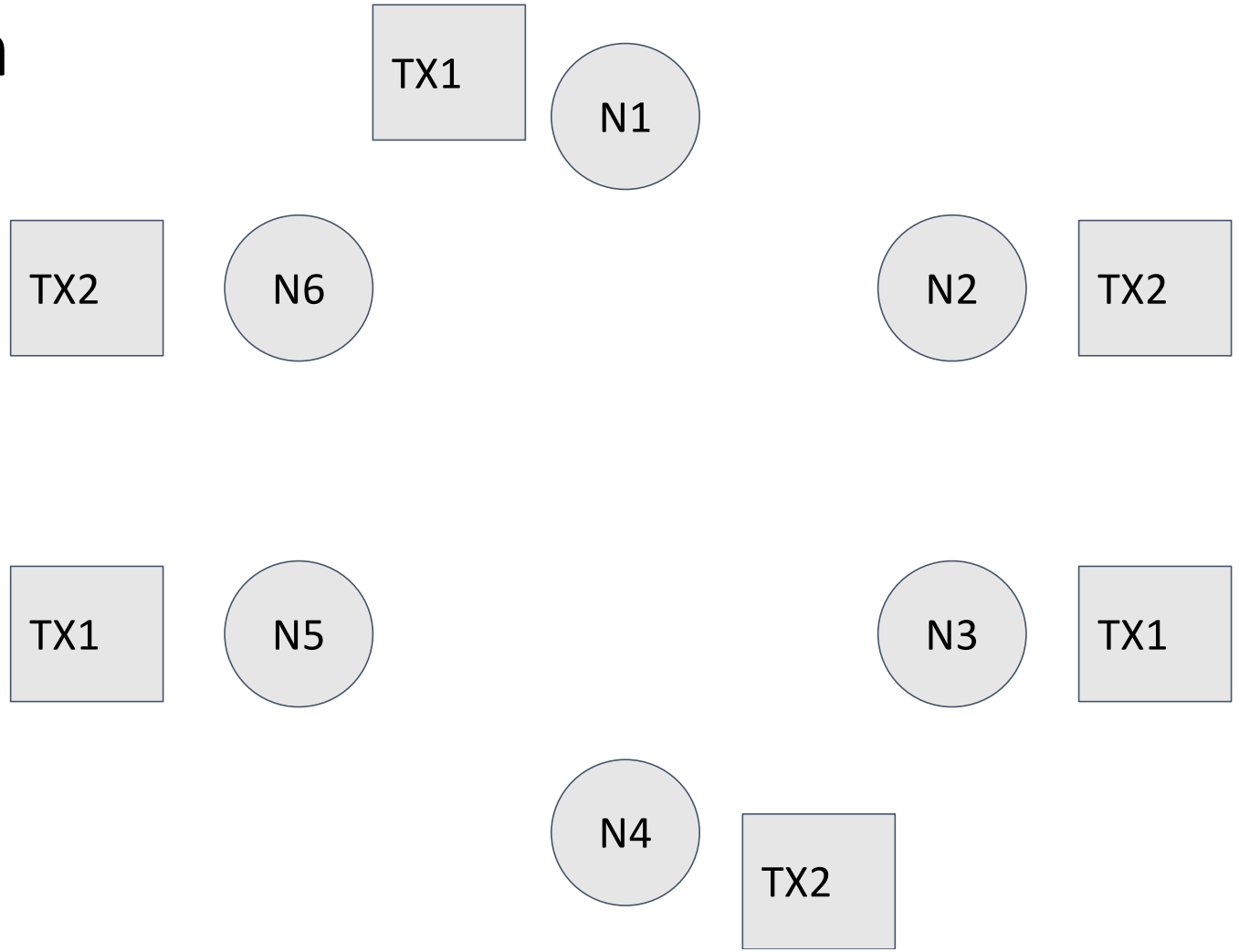
Bitcoin



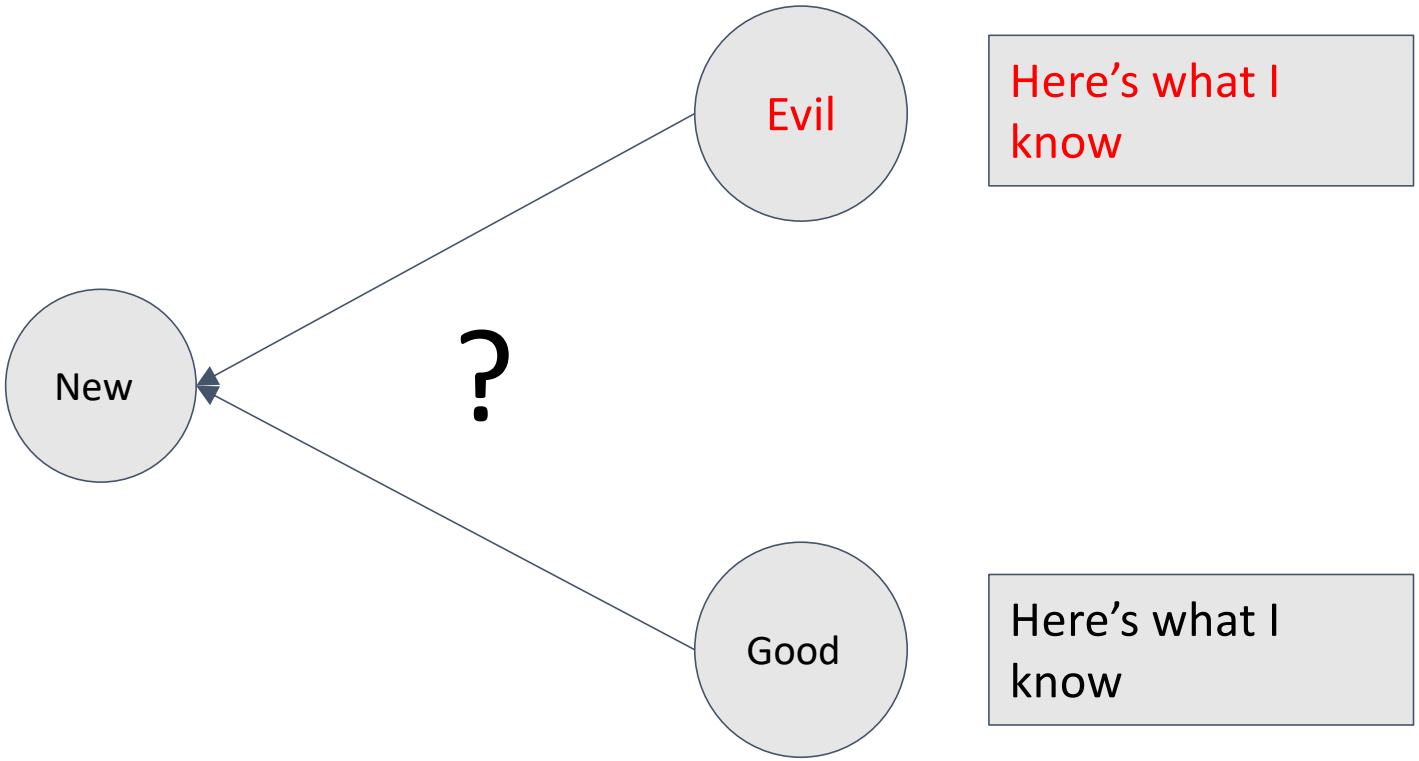
Bitcoin



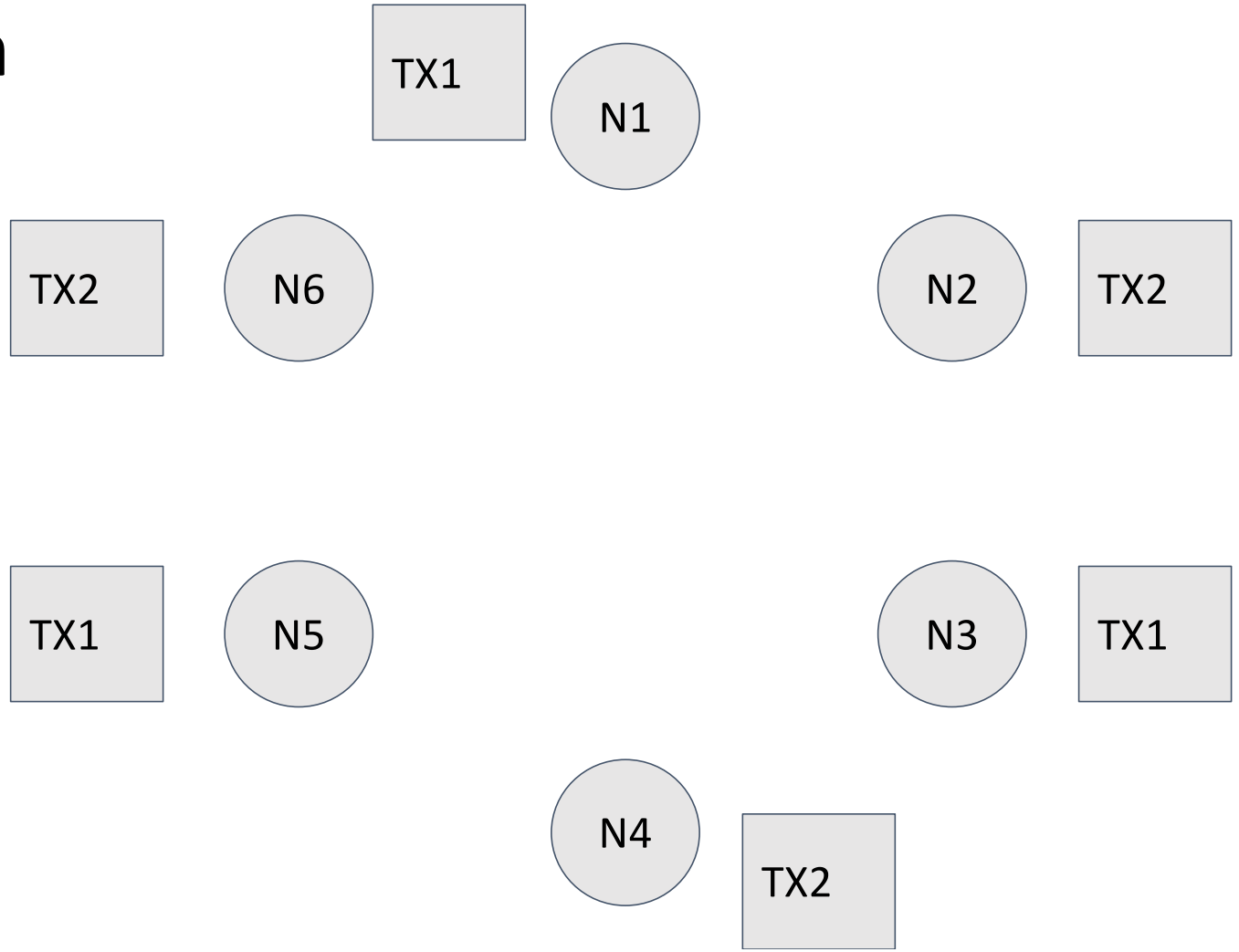
Bitcoin



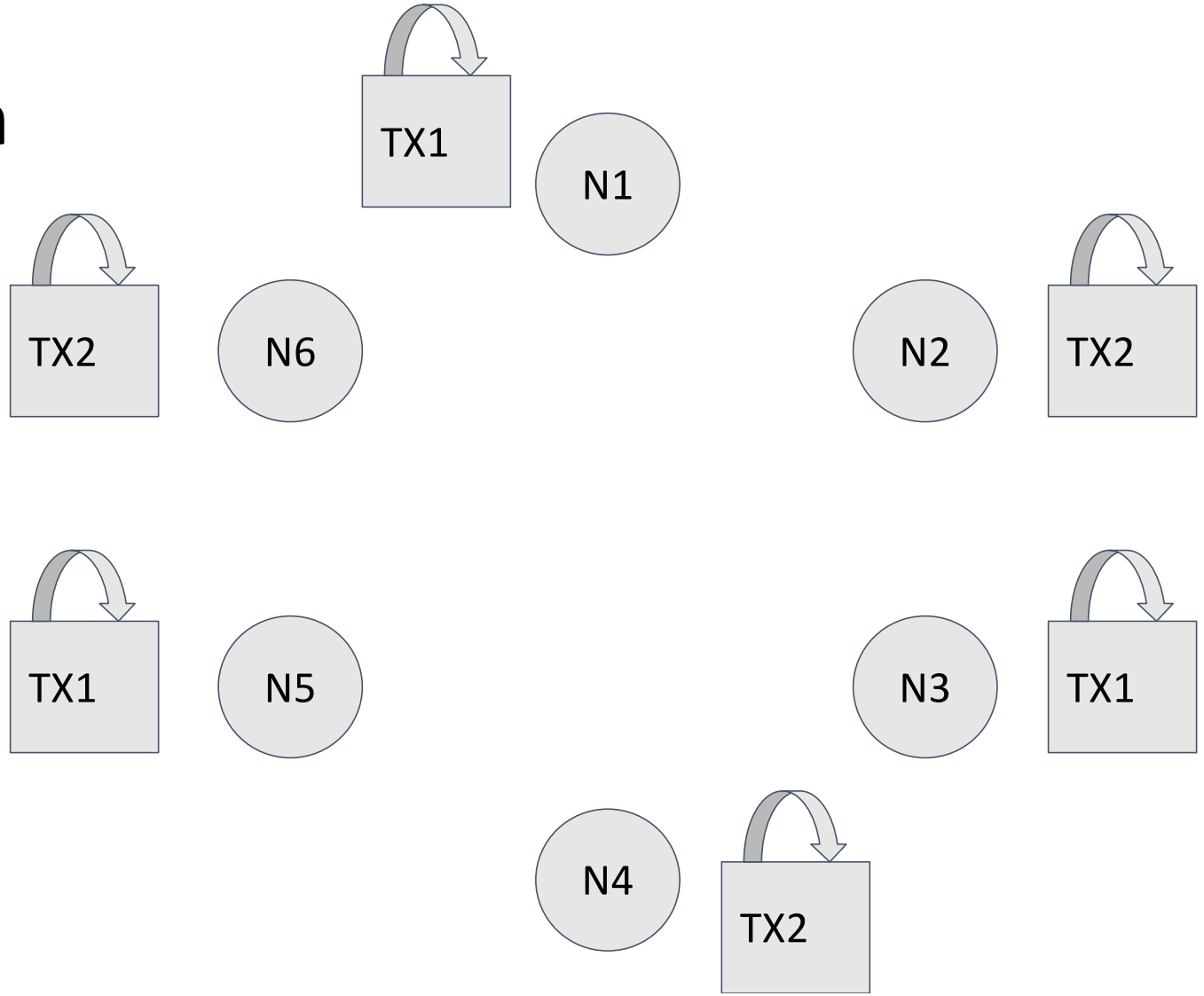
Bitcoin



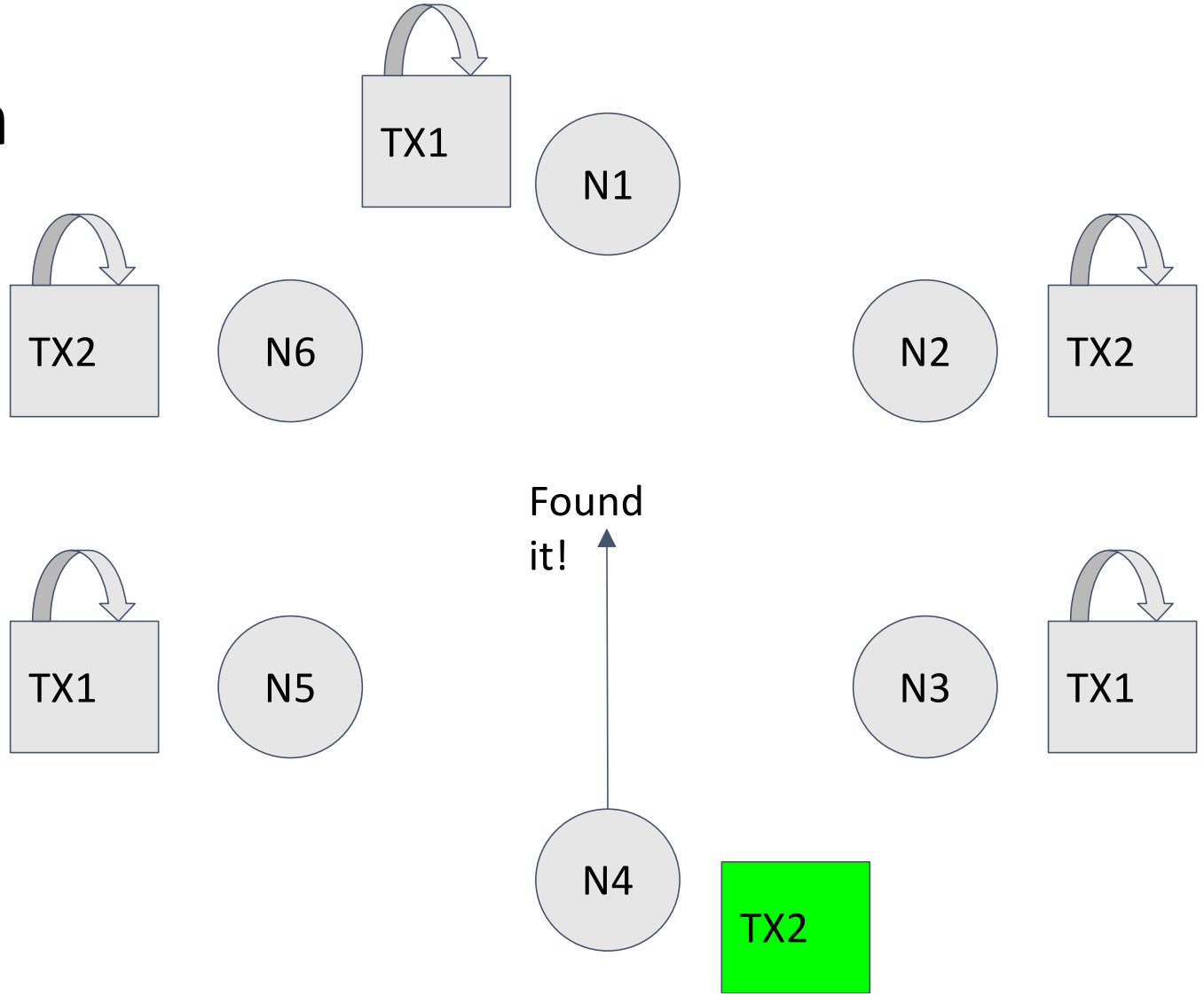
Bitcoin



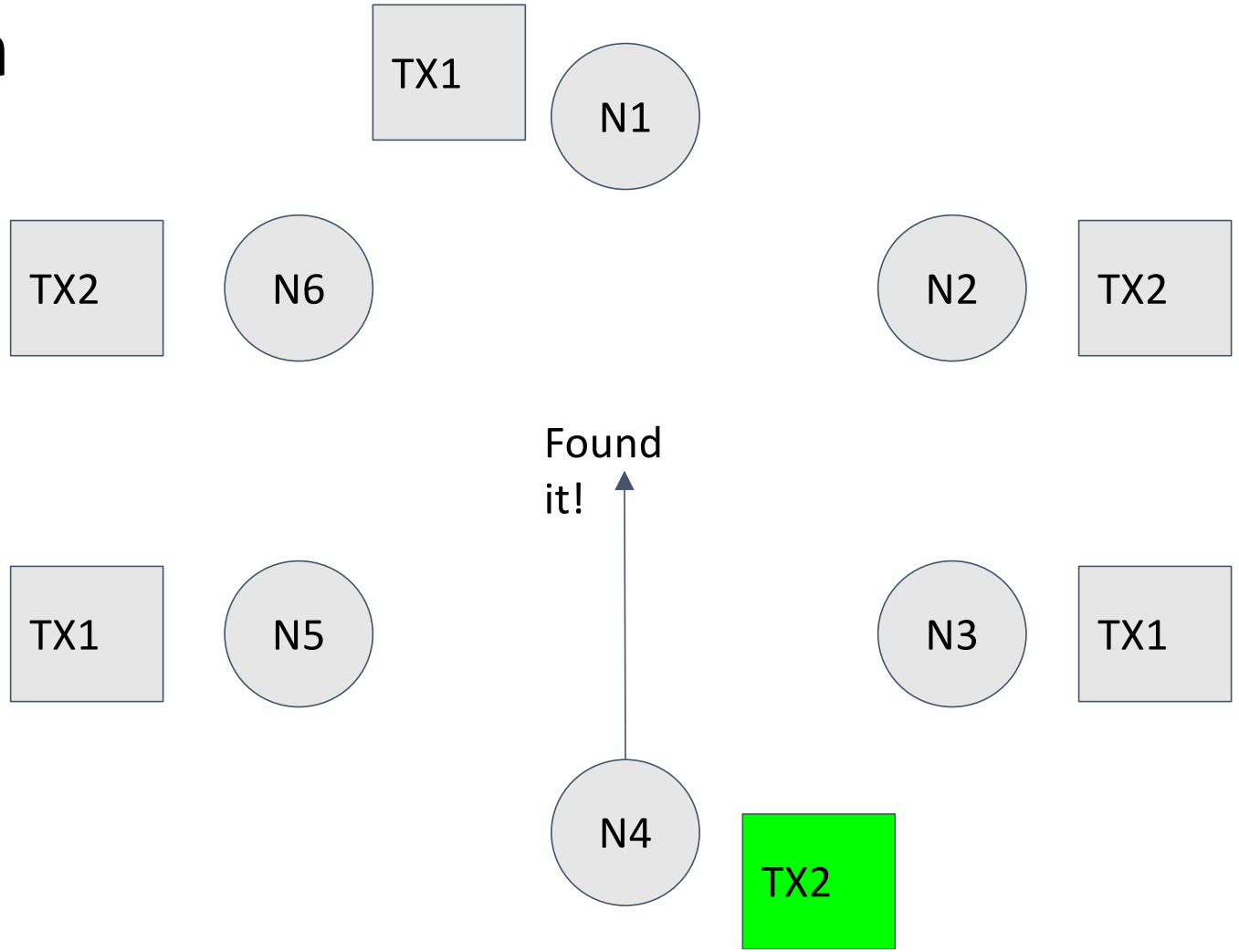
Bitcoin



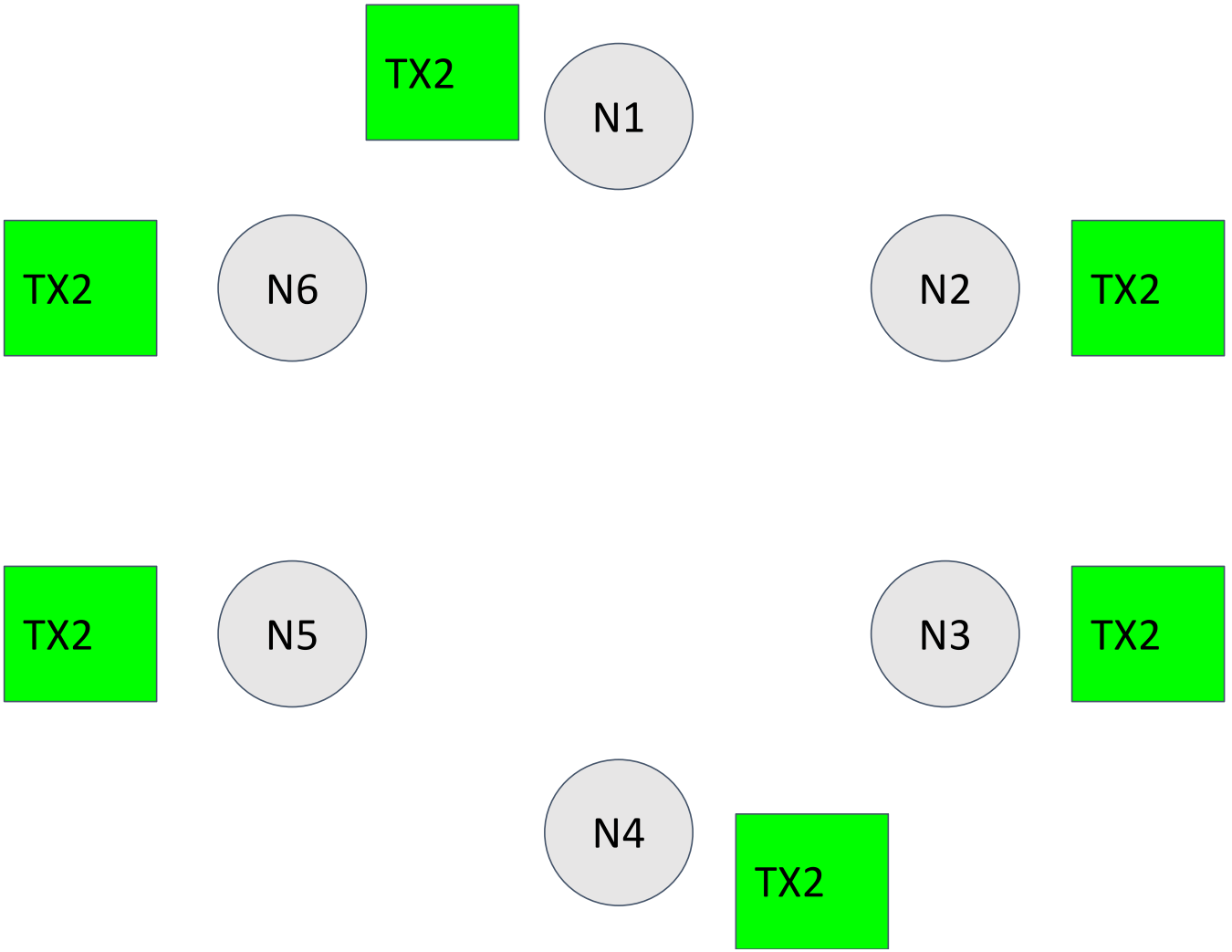
Bitcoin



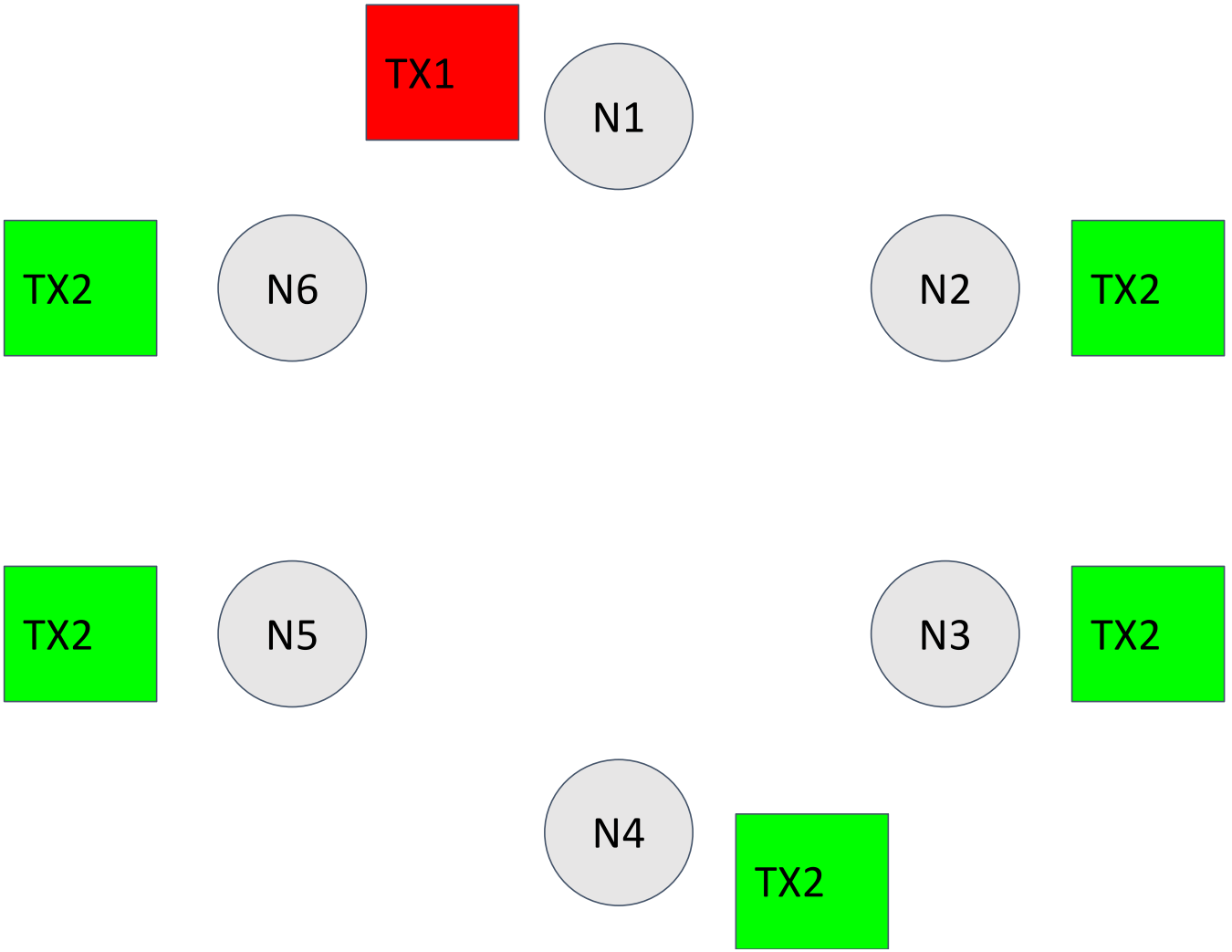
Bitcoin



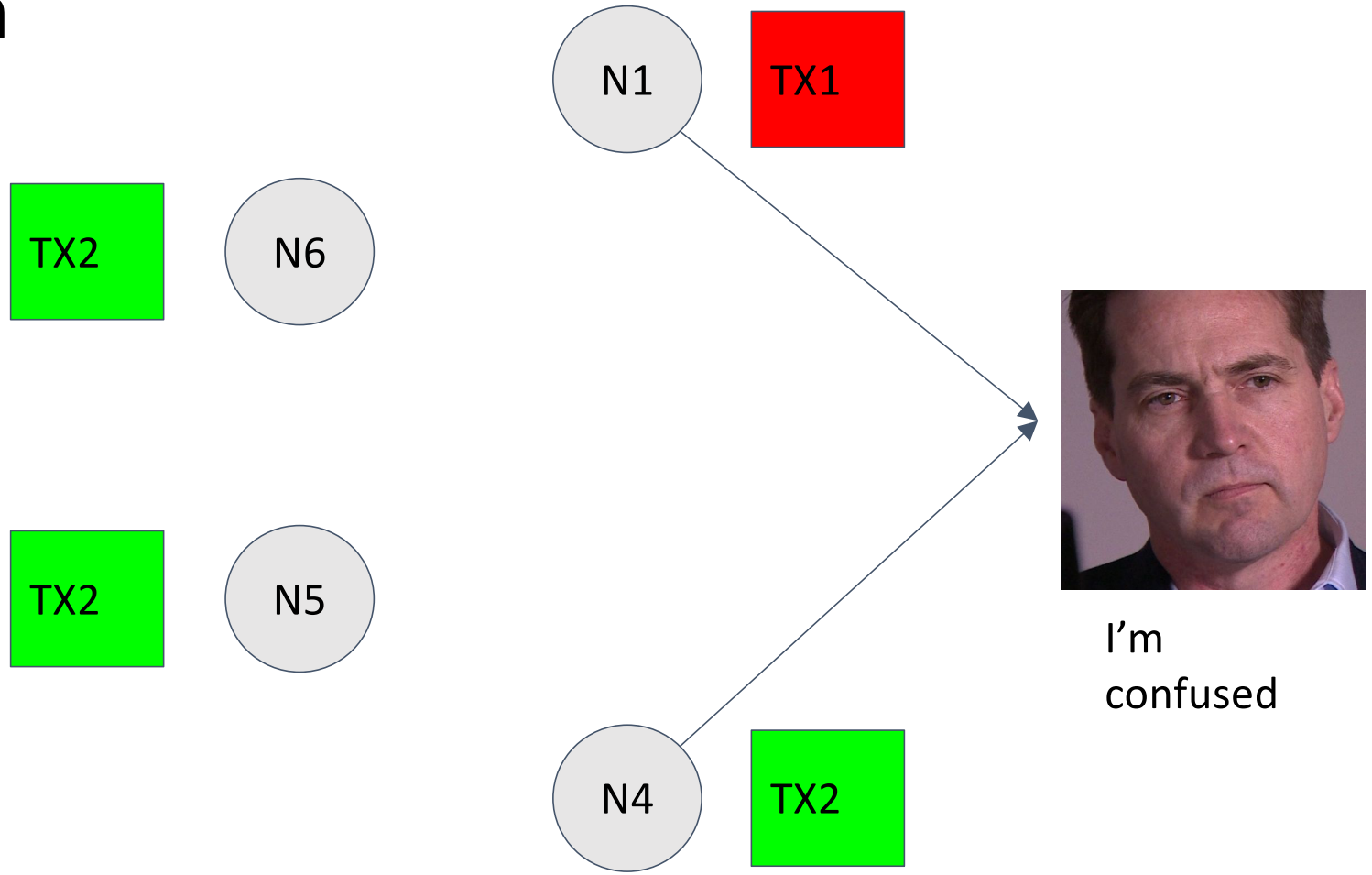
Bitcoin



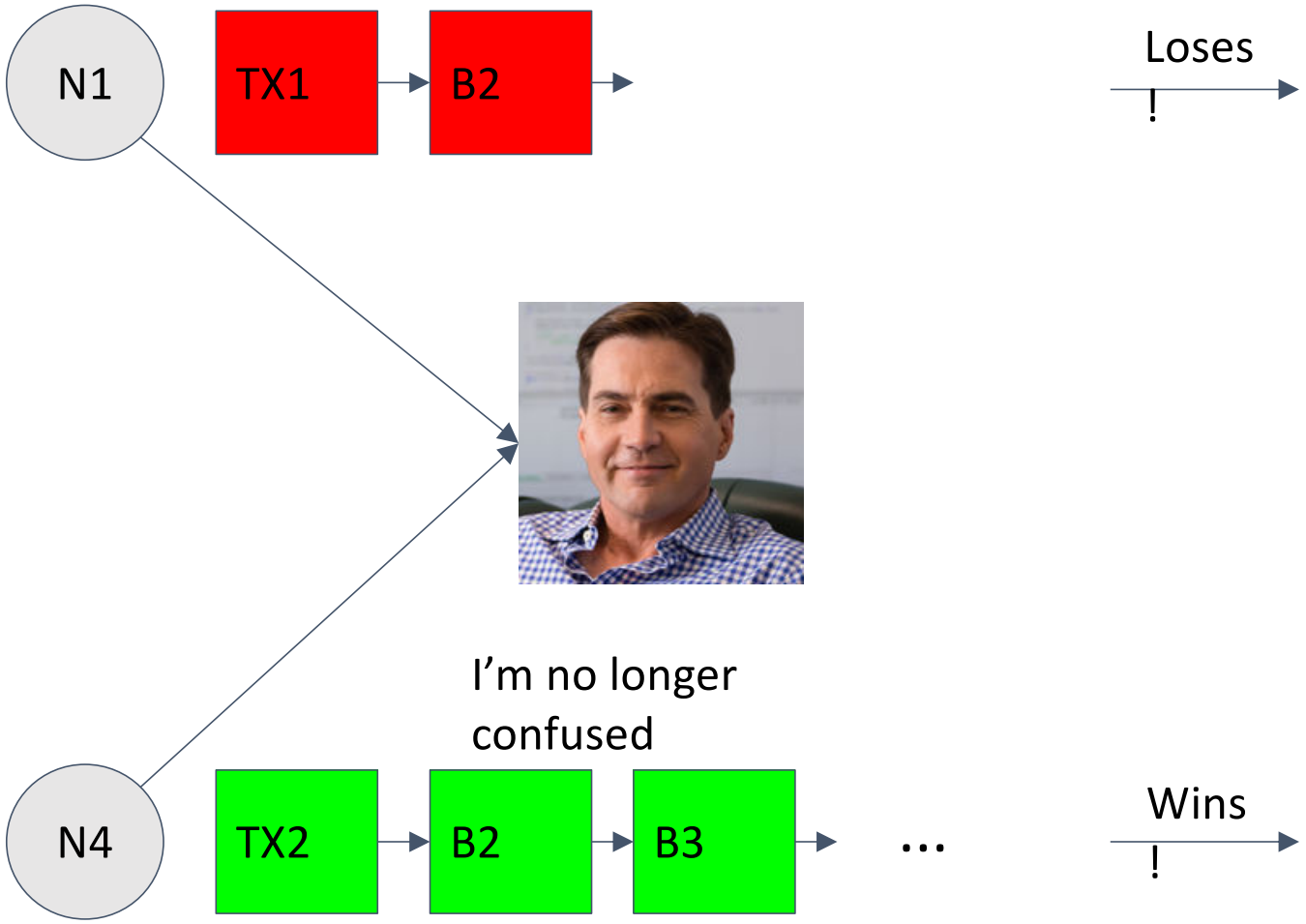
Bitcoin



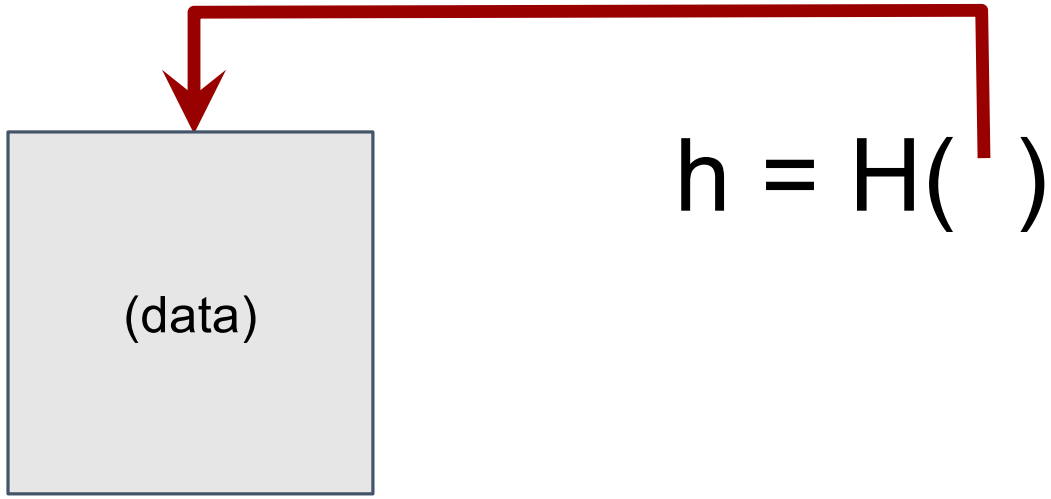
Bitcoin



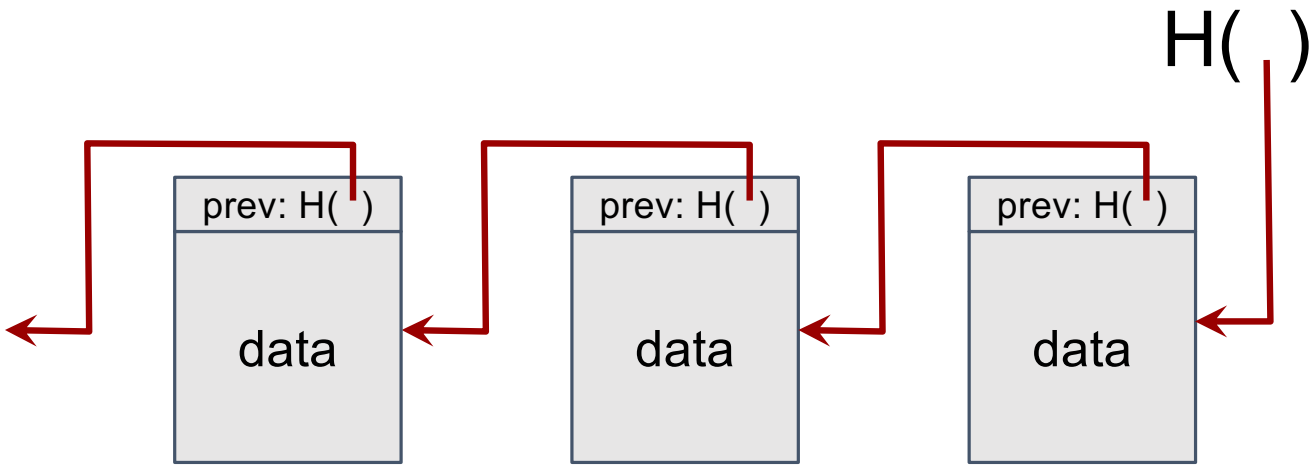
Bitcoin



Hash Pointers

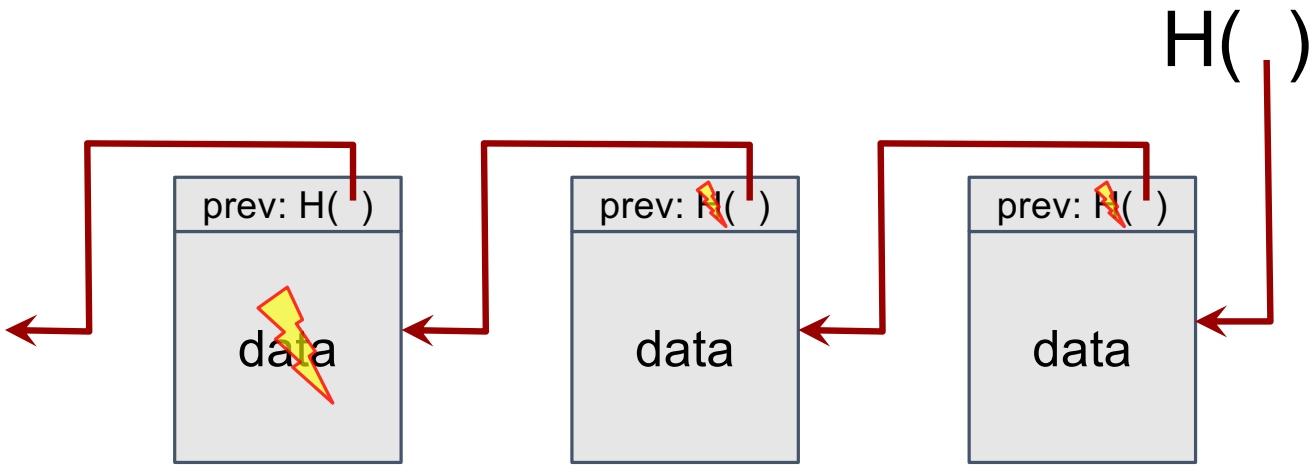


Hash chains



Creates a “tamper-evident” log of data

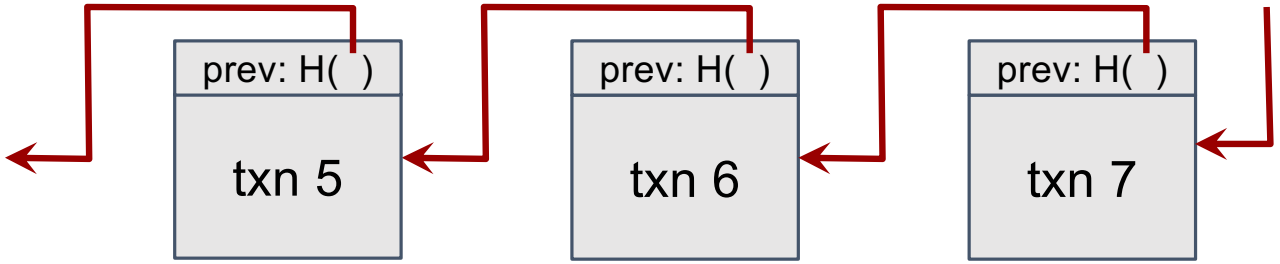
Hash chains



If data changes, all subsequent hash pointers change

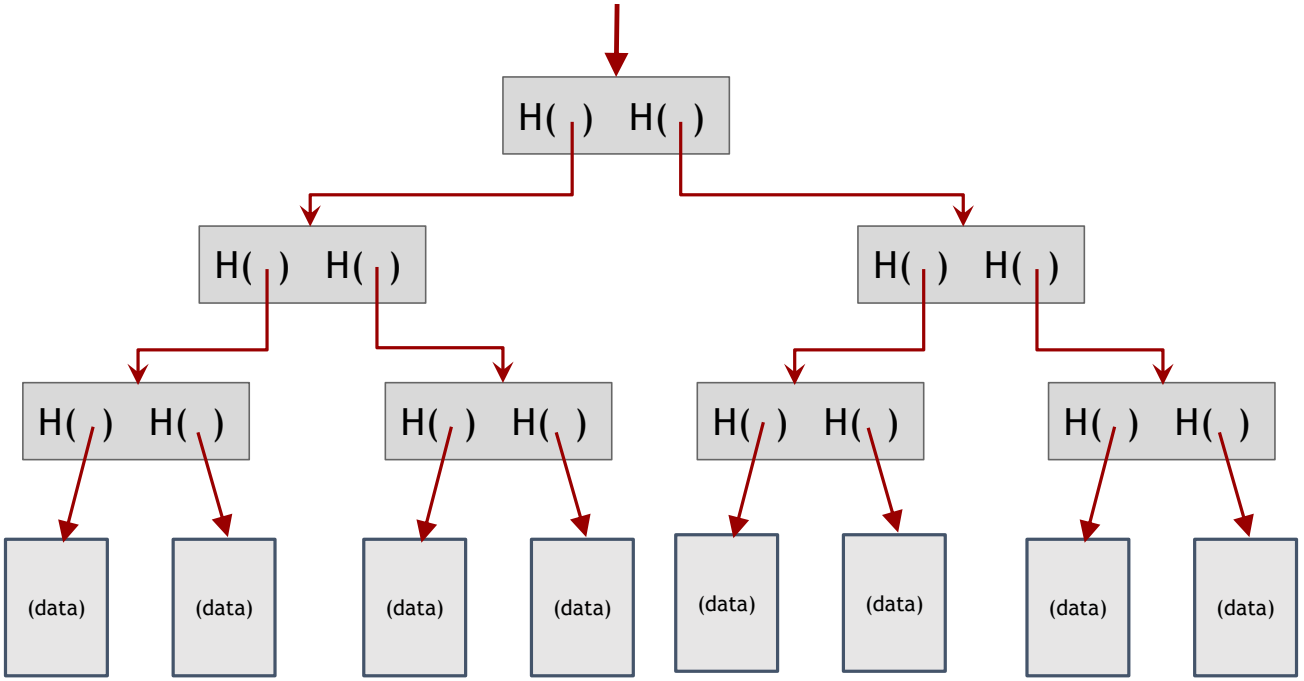
Otherwise, found a hash collision!

Blockchain: Append-only hash chain



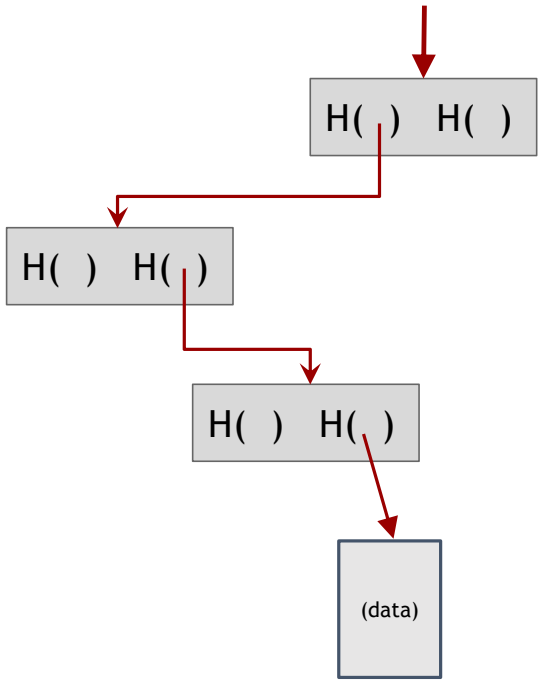
- Hash chain creates “tamper-evident” log of txns
- Security based on collision-resistance of hash function
  - Given m and h = hash(m), difficult to find m’ such that h = hash(m’) and m != m’

Merkle tree



Binary tree with hash pointers

Proving membership in a Merkle tree



show O(log n) items

Can test if transaction is part of tree quickly

What Is Blockchain?

- Shared ledger of transactions in a community
- Transactions are:
  - Authenticated – only the owner of some given asset can transfer part or all of it to someone else
  - Unforgeable – no one can create fake transactions without others noticing
  - Decentralized – no single root of trust, majority decides
  - Consensus-based, single copy – branches may exist for a short time but then they converge to a single copy
- Good for:
  - Keeping track of any kind of assets, transactions and contracts
  - Double spending visible through multiple transactions on same coin

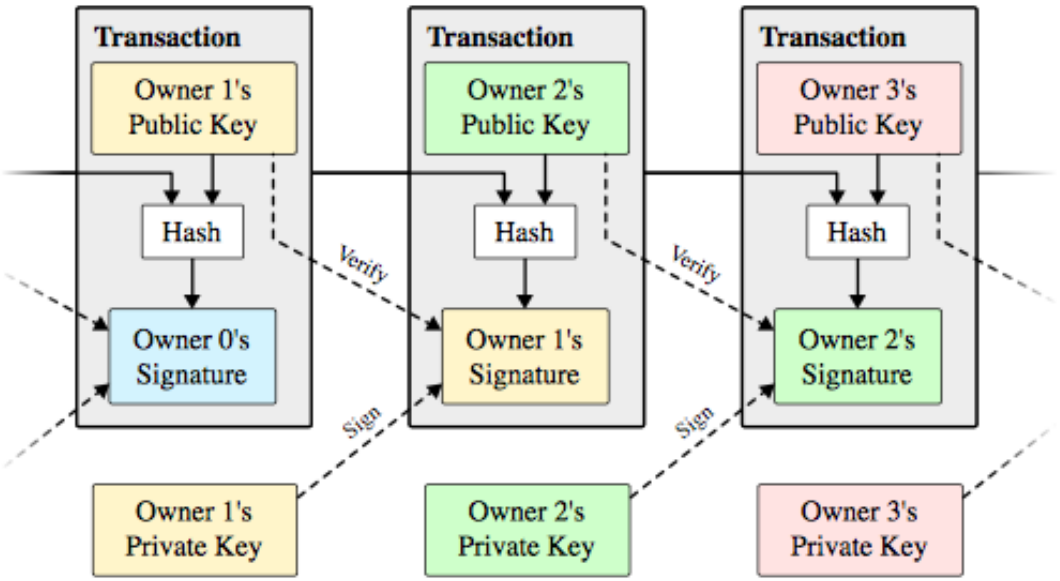
Bitcoin Address

- To receive money, you tell your wallet to generate an “address”
  - This causes the wallet to generate a public-key/secret-key pair
  - The public key is hashed and published as your “address”
    - Why is it hashed? No good reason, really
  - You publish your address
    - Or just tell the payer your address
    - Why no CA to bind address to your identity?

Receiving Money

- Suppose I want to pay you 1 BTC
- I need your address
  - You generated it as in the previous slide
- I generate a "transaction" record and sign it
  - Contains the amount, some metainfo, and your address
    - Also has hash of previous transaction that granted me the money I'm using
  - Signed by my secret key
    - If I lose the secret key associated to the transaction that granted me the bitcoins I'm sending you, I lose that money!

Sequence of transactions



Transaction Record

```
1. {"hash":"7c4025...",                    hash of all following
2. "ver":1,
3. "vin_sz":1,
4. "vout_sz":1,
5. "lock_time":0,
6. "size":224,
7. "in":[
8.   {"prev_out":
9.     {"hash":"2007ae...",          hash of previous transaction
10.   "n":0},
11.   "scriptSig":"304502... 042b2d..."}],
12. "out":[
13.   {"value":"0.31900000",               0.319 BTC being sent
14.   "scriptPubKey":"OP_DUP OP_HASH160 a7db6f OP_EQUALVERIFY OP_CHECKSIG"}]}
  a7db6f is intended recipient
```

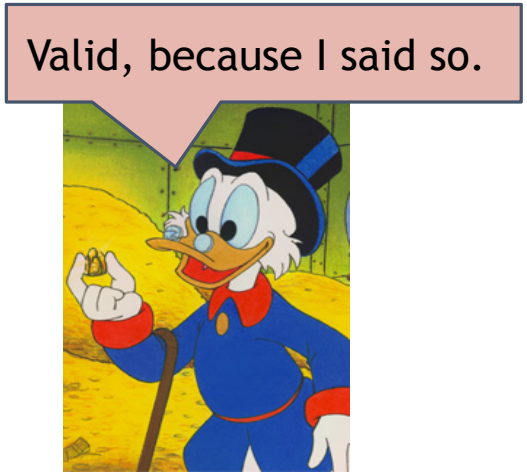
PayCoins transaction consumes (and destroys) some coins, and creates new coins of the same total value

transID: 73 type:PayCoins		
consumed coinIDs: 68(1), 42(0), 72(3)		
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...
signatures		

Valid if:
-- consumed coins valid,
-- not already consumed,
-- total value out = total value in, and
-- signed by owners of all consumed coins

Merges multiple payment coins into one new coin
Returns change from overpayment as new coin

CreateCoins transaction creates new coins



transID: 73	type:CreateCoins	
coins created		
num	*value*	*recipient*
0	3.2	0x...
1	1.4	0x...
2	7.1	0x...

coinID 73(0)

coinID 73(1)

coinID 73(2)

Verifying Transactions

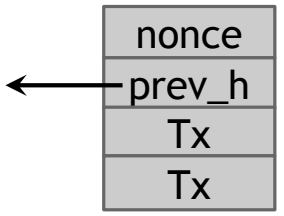
- Why not just check to see if I properly signed the transaction record?
  - I could be cheating!
    - Maybe I don't own the coins I'm sending
    - Maybe I already spent those coins with someone else
- So instead the "bitcoin network" verifies the transaction
  - This is hard-by-design because there is a nice payoff for doing it
  - It also means a cheater would have to have more computing power than the rest of the network

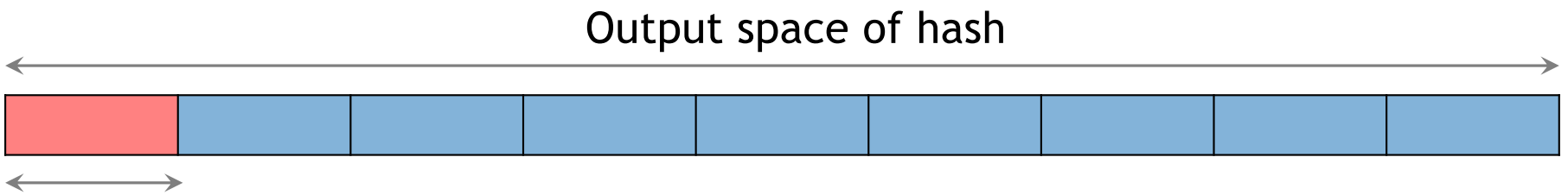
The Blockchain

- Every verifier (or “miner”) on the network has an entire history of all transactions
  - Called the “blockchain”
- This is a chain of transactions that tracks where each bitcoin has been
  - Every transaction has the hash of the previous transaction that granted the coins
  - Once a transaction has been verified, it is added to the blockchain by all nodes of the network
- Goal: prevent evildoer(s) from verifying bad transactions
  - Solution: make it expensive by requiring lots of computations, distribute task of verification

Hash puzzles

To create block, find nonce s.t.
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) is very small





If hash function is secure:
only way to succeed is to try enough nonces until you get lucky

How to Mine Bitcoins

- Suppose you want to verify a transaction. Must find nonce such that N leading digits of hash are zero

  - Suppose the transaction is “hello”
  - Compute SHA256(“hello:0”)
    - `a61bb398117fe…`
  - Compute SHA256(“hello:1”)
    - `61b7a90017562…`
  - …
  - Compute SHA256(“hello:917712”)
    - `0000718a5dce3…` `winner!`

How hard is this?

- To get a leading 0 digit in hex, assuming SHA256 is random
  - 1/16 chance
  - 16 expected trials
- Two leading 0's
  - 256 expected trials
- In reality, to verify a bitcoin transaction you have to get below the target
  - This should take about 10 mins given the power of the network
  - This is recalibrated every 2 weeks

PoW property 1: difficult to compute

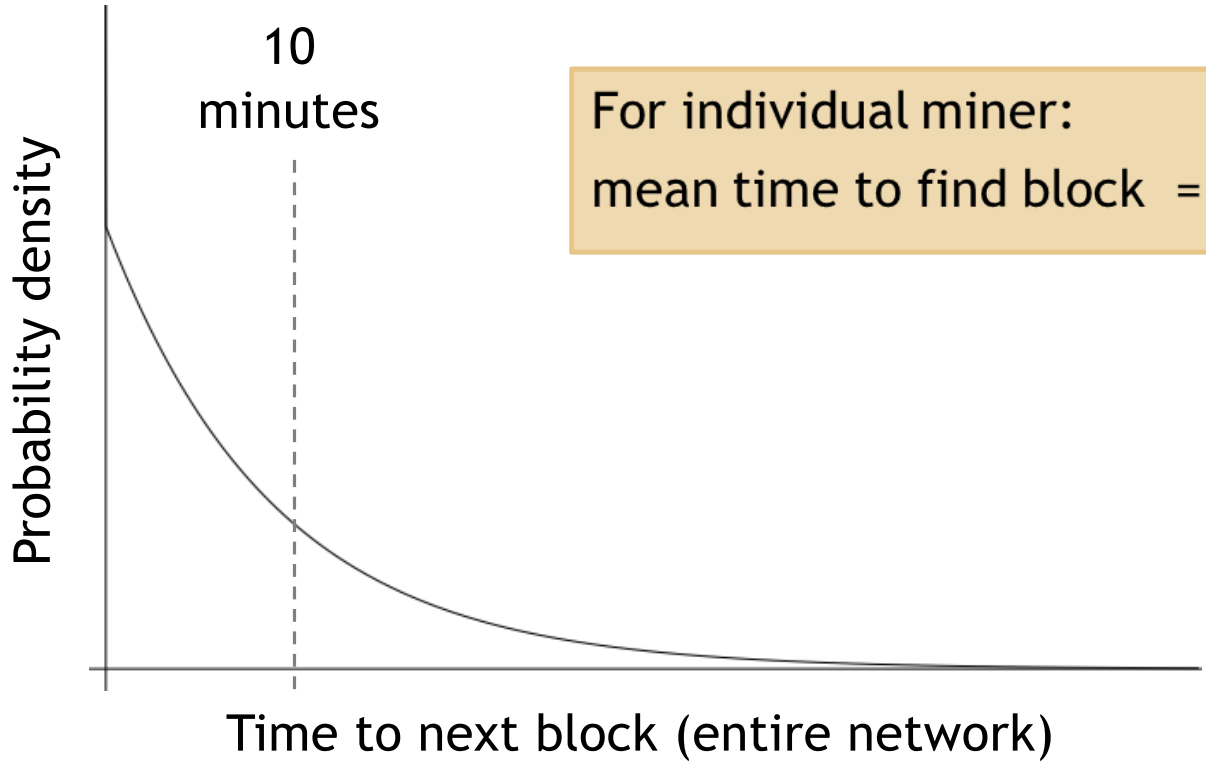
As of Aug 2014: about 10^{20} hashes/block

Only some nodes bother to compete — miners

Key security assumption

Attacks infeasible if majority of miners weighted by hash power follow the protocol

Solving hash puzzles is probabilistic



For individual miner:

$$\text{mean time to find block} = \frac{10 \text{ minutes}}{\text{fraction of hash power}}$$

PoW property 2: parameterizable cost

Nodes automatically re-calculate the target every two weeks: more leading zeroes

Goal: average time between blocks = 10 minutes

Prob (Alice wins next block) =
fraction of global hash power she controls

PoW property 3: trivial to verify

Nonce must be published as part of block

Other miners simply verify that
H(nonce ‖ prev_hash ‖ tx ‖ … ‖ tx) < target

Mining economics

If mining reward (block reward + Tx fees)	>	hardware + electricity cost	→	Profit

Complications:

- fixed vs. variable costs
- reward depends on global hash rate

Aspects of decentralization in Bitcoin

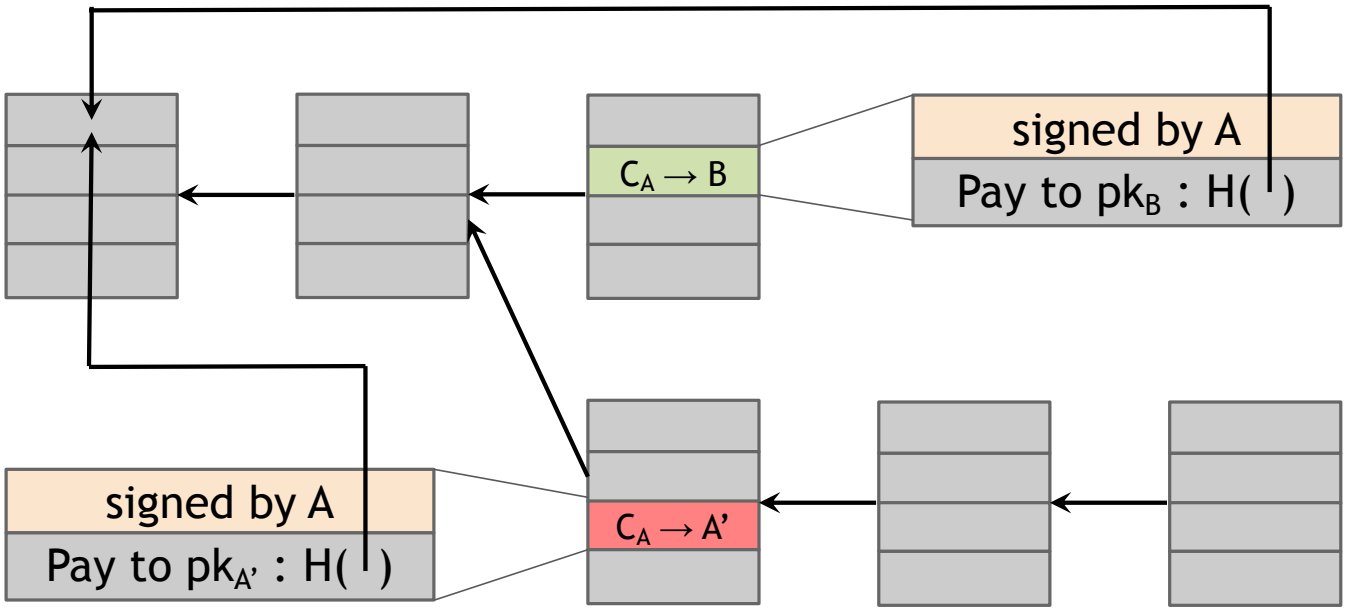
1. Who maintains the ledger?
2. Who has authority over which transactions are valid?
3. Who creates new bitcoins?
4. Who determines how the rules of the system change?
5. How do bitcoins acquire exchange value?

Beyond the protocol:
exchanges, wallet software, service providers...

Announcing Verification

- Once an entity verifies the transaction it broadcasts it to the network
  - The other nodes stop trying to compete
    - Because they already lost
  - Other nodes verify that the transaction is valid
    - Money spent was legitimately owned
    - Verifier got below the target
  - Then transaction is added to the end of the blockchain

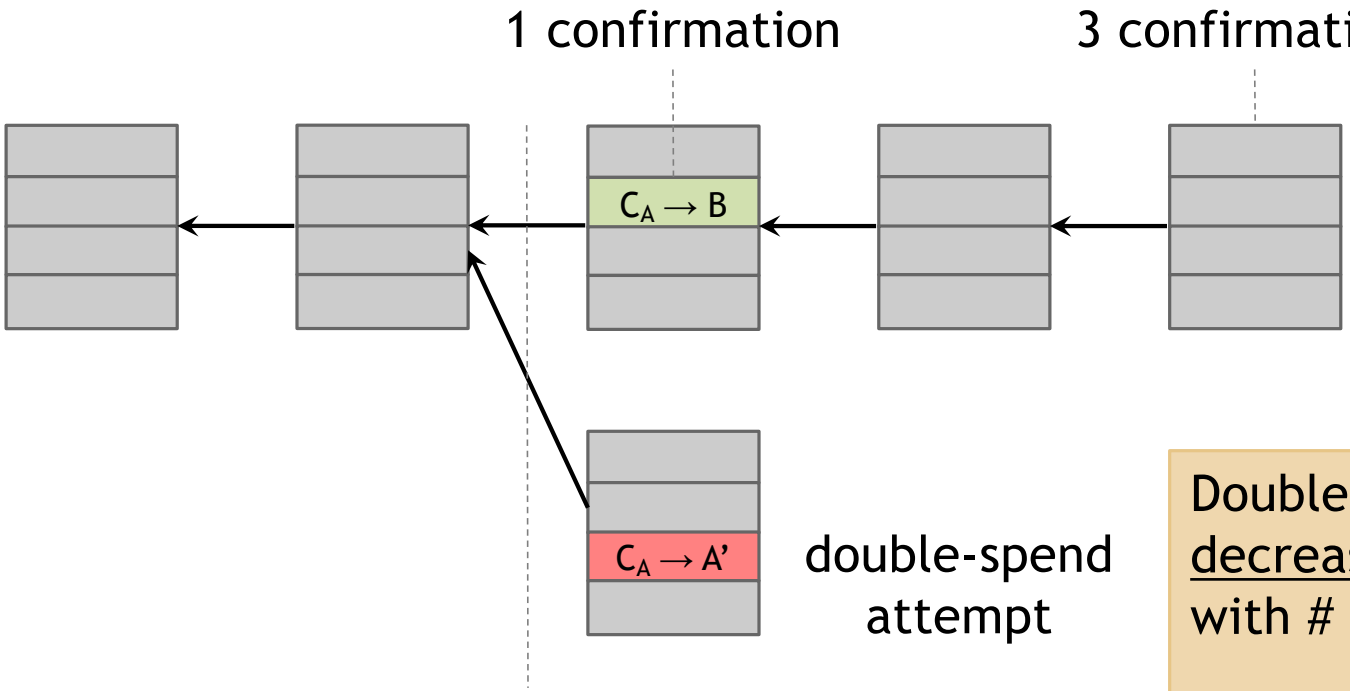
What can a malicious node do?



Double-spending attack

Honest nodes will extend the longest valid branch

From Bob the merchant's point of view

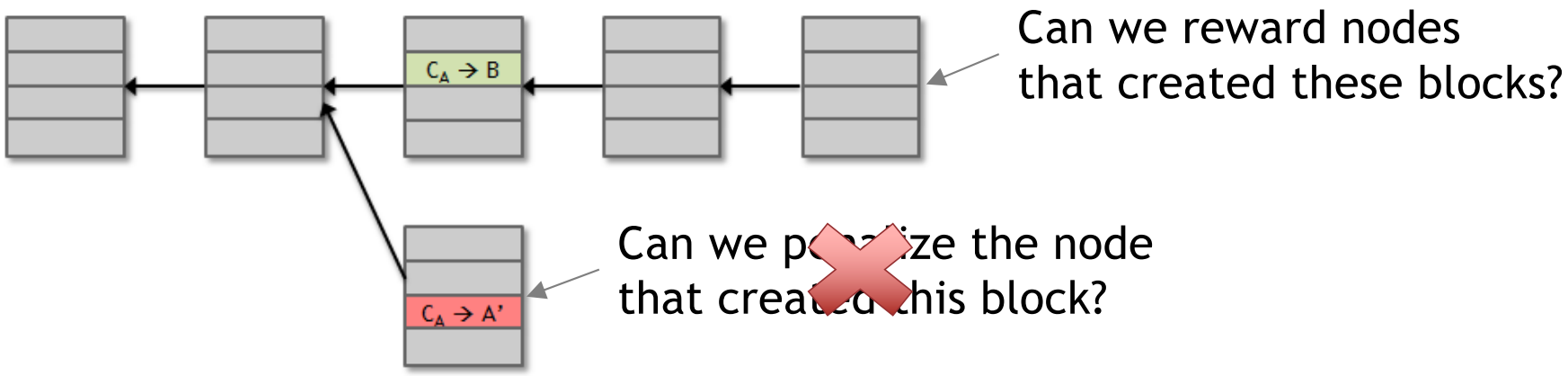


Double-spend probability <u>decreases exponentially</u> with # of confirmations

Most common heuristic:
6 confirmations

Assumption of honesty is problematic

Can we give nodes incentives for behaving honestly?



Everything so far is just a distributed consensus protocol
But now we utilize the fact that the currency has value

Incentive 1: block reward

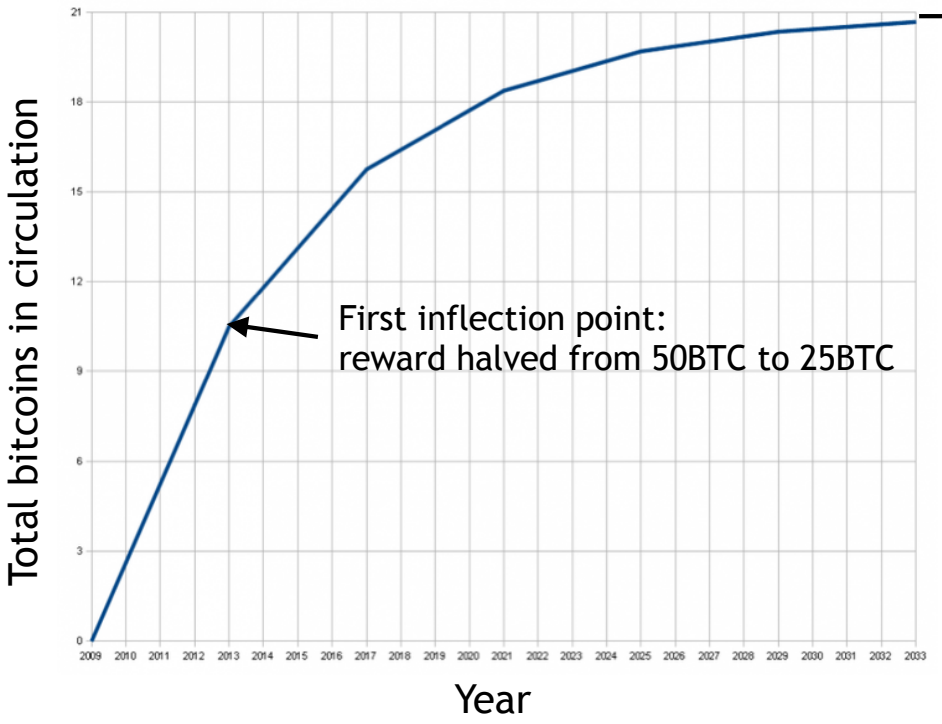
Creator of block gets to

- include special coin-creation transaction in the block
- choose recipient address of this transaction

Value varies: was 25 BTC, then halved to 12.5

Block creator gets to “collect” the reward only if the block ends up on long-term consensus branch!

There’s a finite supply of bitcoins



Total supply: 21 million

Block reward is how new bitcoins are created

Runs out in 2040. No new bitcoins unless rules change

Incentive 2: transaction fees

Creator of transaction can choose to make output value less than input value

Remainder is a transaction fee and goes to block creator

Purely voluntary, like a tip – but miner may only accept transactions with a TX fee

What can a “51% attacker” do?

Steal coins from existing address? ✗

Suppress some transactions?

- From the block chain ✓
- From the P2P network ✗

Change the block reward? ✗

Destroy confidence in Bitcoin? ✓✓

Ethereum

- Like bitcoin
  - Proof-of-work
  - Hash chains
- Adds smart contracts
  - Executed automatically by a transaction

Example: bet on an event

```
if HAS_EVENT_X_HAPPENED() is true:
     send(party_A, 1000)
else:
     send(party_B, 1000)
```

How Ethereum Works

- Two types of accounts:
  - **Normal account** like in Bitcoin
    - has balance and address
  - **Smart Contract account**
    - like an object: containing (i) code, and (ii) private storage (key-value storage)
    - Code can
      - Send ETH to other accounts
      - Read/write storage
      - Call (ie. start execution in) other contracts

DNS: The “Hello World” of Ethereum

```
data domains[](owner, ip)
```

Private Storage

```
def register(addr):
    if not self.domains[addr].owner:
        self.domains[addr].owner = msg.sender
```

Can be invoked by other accounts

```
def set_ip(addr, ip):
    if self.domains[addr].owner == msg.sender:
        self.domains[addr].ip = ip
```

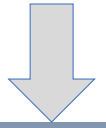
Example

```
1  contract Greetings {
2      string greeting;
3      function Greetings (string _greeting) public {
4          greeting = _greeting;
5      }
6
7      /* main function */
8      function greet() constant returns (string) {
9          return greeting;
10     }
11 }
```

What you write

What other see on the blockchain

```
6060604052604051610250380380610250833981016040528.....
```



```
PUSH 60
    PUSH 40
    MSTORE
    PUSH 0
    CALLDATALOAD
.....
```

What people get from the disassembler

Transactions in Ethereum

- Normal transactions like Bitcoin transactions
  - Send tokens between accounts
- Transactions to contracts
  - like function calls to objects
  - specify which object you are talking to, which function, and what data (if possible)
- Transactions to create contracts

Blockchain State

Bitcoin's state consists of key value mapping addresses to account balance

Address	Balance (BTC)
0x123456…	10
0x1a2b3f…	1
0xab123d…	1.1

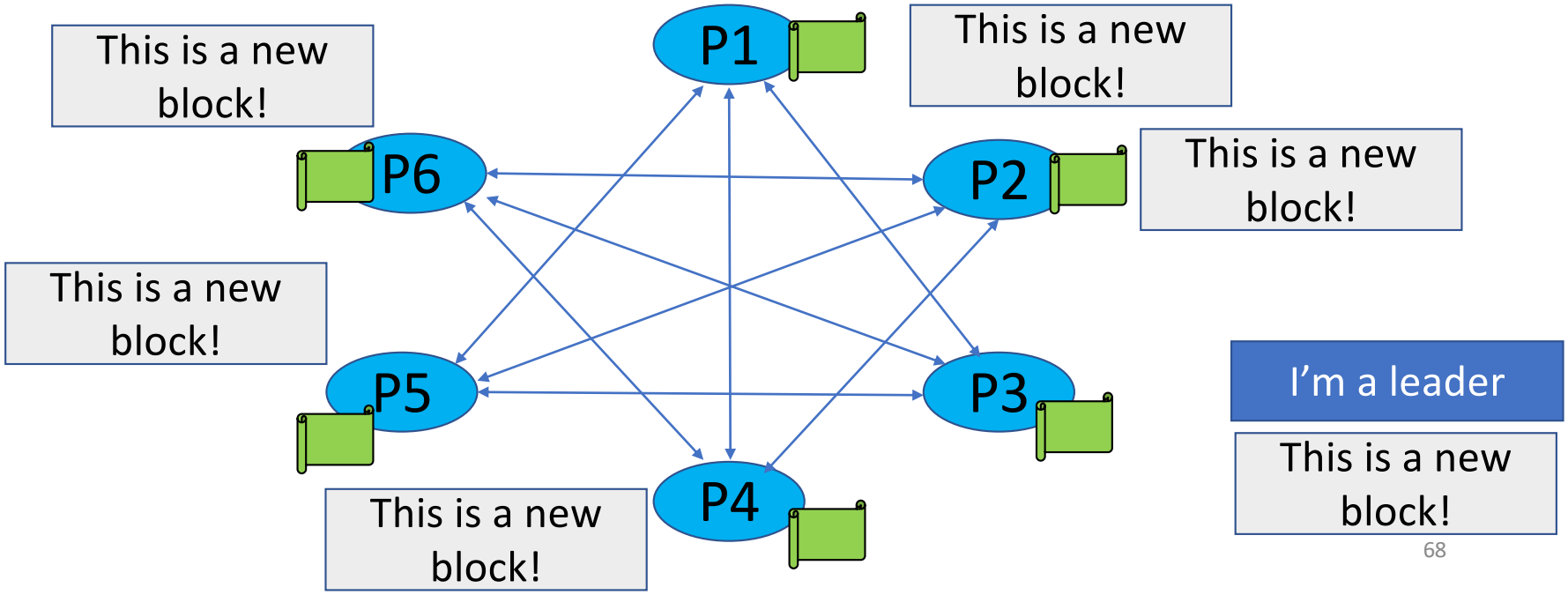
Ethereum's state consists of key value mapping addresses to account objects

Address	Object
0x123456…	X
0x1a2b3f…	Y
0xab123d…	Z

Blockchain != Blockchain State

Code execution

- Every (full) node on the blockchain processes every transaction and stores the entire state



Dos Attack Vector

- Halting problem
  - Cannot tell whether or not a program will run infinitely
  - A malicious miner can DoS attack full nodes by including lots of computation in their txs
    - Full nodes attacked when verifying the block

```
uint i = 1;
while (i++ > 0) {
    donothing();
}
```

Solution: Gas

- Charge fee per computational step (“gas”)
  - Special gas fees for operations that take up storage

Operation	Gas	GasCost
PUSH1	111741	3
PUSH1	111738	3
MSTORE	111726	12
CALLDATASIZE	111724	2
ISZERO	111721	3
PUSH2	111718	3
JUMPI	111708	10

Sender has to pay for the gas

- **`gasprice`**: amount of ether per unit gas
- **`startgas`**: maximum gas consumable
  - If **startgas** is less than needed
    - Out of gas exception, revert the state as if the TX has never happened
    - Sender still pays all the gas
- TX fee = gasprice * consumedgas
- Gas limit: similar to block size limit in Bitcoin
  - Total gas spent by all transactions in a block < Gas Limit